キャンパス内コミュニケーション促進のための P2Pチャットソフトの開発

福原　直*　森口一郎**

組織内に設置されているコンピュータとLAN（Local Area Network）を使い、LAN内にいる人間同士のコミュニケーションを円滑にするソフトウェアを開発し、実証した。距離的に身近な人とのコミュニケーションを目的とし、外部ネットワークとはあえて通信をさせないことにより、マルチキャストを使って他コンピュータを検索することができる。検索後は他コンピュータとP2P（Peer to Peer）により通信を行うことで、サーバを使わずに他コンピュータとの通信を可能にし、ユーザにとって簡単にネットワークを構築でき、気軽にコミュニケーションできるソフトウェアを提案する。

キーワード：TCP/IP，UDP，マルチキャスト，ソケット，P2P（Peer to Peer），OpenSSL，公開鍵暗号方式，共通鍵暗号方式，RSA，Blowfish，トリップ，WindowsAPI，WinSock

Development of a P2P Chat Software Promoting Communication in Campuses

Ataru FUKUHARA and Ichirou MORIGUCHI

We have developed the software that is able to extend people's communication by local area networks and computers in any organizations. In local area networks, this software searches other computers by multicast routing. Since the main purpose of this software is to communicate with other users in short distance, there is no need to communicate with outside networks. After searching, this software communicates with other computers by peer-to-peer communication architectures. This software is easy for users to construct communication networks because any server computer is not necessary. We propose this software by which every people can easily communicate each other.

Keyword：TCP/IP, UDP, Multicast, Socket, Peer to Peer, OpenSSL, Public key cryptosystem, Common key cryptosystem, RSA, Blowfish, Trip, WindowsAPI, WinSock

*東京情報大学総合情報学部情報システム学科
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems
2009年4月よりSONY株式会社に所属
**東京情報大学総合情報学部情報システム学科
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems

2009年8月19日受理

1．はじめに

大学等の巨大な組織内には、他人とコミュニケーションをとることが得意な人間もいれば、そうでない人間もいる。しかし、コミュニケーションをとることが苦手な人でも、コンピュータとネットワークを用いたチャット等のソフトウェアならば、匿名性があるため気軽にコミュニケーションをとることができるであろう。このようなネットワークを介したコミュニケーションで物理的に身近にいる人間と交流を深めることができれば、次のステップとして実際に会ったときのコミュニケーションも円滑にできる可能性が高いと思われる。

しかしながら、誰かが用意したチャットサーバを使うことは、そのサーバの管理者による監視の可能性が排除できず、また、パケット盗聴による第三者への情報流出の危険もある。さらに、サーバを設置するにしても手間と物理的資源、そして設置する者のスキルが必要である。これらは結果として、現実世界でのコミュニケーションに恐怖心を抱いているユーザの積極的かつ自由なネット上でのコミュニケーションを抑止してしまうであろう。よって、これらの状況を踏まえ、本研究では以下の機能を備えたソフトの開発を目標とした。

(1) サーバを必要とせず、ユーザのPC同士がコミュニケーションネットワークを自動的に構築できること。
(2) コミュニケーション内容を秘匿でき、匿名性を保障することによって教員や職員による干渉を排除できること。
(3) 複数の参加者によるチャットだけではなく、1対1のコミュニケーションもサポートすること。
(4) 単なるチャットだけではなく、画像ファイルや授業資料などのファイル送受信もできること。
(5) 使用方法を知らないユーザが使っても容易にコミュニケーションネットワークに参加できるような、容易なインタフェースを提供すること。

(1) に関してはP2P(Peer to Peer)ネットワークモデルで解決できるように思えるが、実際は既存P2Pソフトは全て、中央サーバが必要なハイブリッドP2P形態か、あるいは、最初に接続すべきノードの情報を何らかの方法で配布しなくてはならないピュアP2P形態である[1]。本研究では初期ノードの配布が不要なピュアP2Pソフトの開発を目指した。

この初期ノード配布が不要なシステムを実現する方法として、本研究ではマルチキャストを使って他のコンピュータを探す手法を提案し、実装することとした。以下にマルチキャストを用いたコミュニケーションネットワーク構築の概要を説明する。

本研究で作成したソフトウェアでは、ユーザが好きな名前を付けたチャットルームをユーザが自由に作ることができる。また、ユーザは自分の興味のある名前のチャットルームに参加することができるので、自分と興味が一致するユーザ同士でコミュニケーションをとることができる。

チャットルームを作ったユーザを本研究ではホストと呼び、チャットルームに入室する際にはホストに接続する必要がある。一般的に、接続に必要なIPアドレス(Internet Protocol Address)とPort番号を知るためにはサーバに問い合わせるという方法が一般的であるが、それではサーバを設置して管理するスキルと手間が必要である。しかし、マルチキャストを使うことでサーバがなくともホストのIPアドレスとPort番号を取得できる。さらにホストに接続後はルーム内のメンバーとP2Pで通信させることで、サーバなしでの通信を可能にした。

サーバなしで仮想コミュニケーションのネットワークを構築できるということは、サーバ設置管理のスキルも手間も不要ということであり、またその組織内のネットワーク管理者にも手間をかけずにすむということである。これは

利用者のみで気軽にネットワークを構築できるという点で有意義である。

しかし、組織内に複数のLAN(Local Area Network)が存在し、そのLAN間でマルチキャストルーティングが設定されていない環境も存在する。この場合、マルチキャストパケットが他のLANに届かず、LANごとに仮想コミュニケーションのネットワークが分断される。そういった条件下でも組織内の全てのLANで１つの仮想コミュニケーションのネットワークを構築するためには、各ノードのIPアドレスとポート番号のみを管理する簡易なサーバソフトウェアを設置管理することで通信を可能にした。

以上のように、誰でもコンピュータとネットワークを用いて気軽に他人とコミュニケーションがとれる設計により本研究で作成したソフトウェアはEveryone's Light-hearted Communication.(みんなの気軽なコミュニケーション)の頭文字をとってEveLiCoと名付けた。

2．Peer to Peerにおけるユーザ識別

2.1　Peer to Peer

Peer to Peer(以後、略称のP2Pと呼ぶ)とはネットワークモデルの１つである。クライアント・サーバモデルのネットワークとは違い、P2Pモデルのネットワークでは全てのコンピュータが対等な関係でネットワークが成り立っている。また、本論文ではP2Pのネットワークに参加しているコンピュータをノード(Node)と呼ぶ。

P2Pには2種類の形態が存在する[1]。１つはハイブリッドP2P(Hybrid P2P)と呼ばれ、ノード情報を管理する中央サーバが存在し、あるノードが他のノード情報を得る時だけ中央サーバに尋ね、実際の通信はノード同士で行う形態である(図1)。もう１つの形態はピュアP2P(Pure P2P)と呼ばれ、中央サーバが存在せず、ノードが他のノード情報を保持することで通信を行う形態である(図2)。しかし、初めて参加したノードは何らかの方法で少なくとも1つの他のノードのアドレスを知る必要があり、全てのピュアP2Pソフトでは「初期接続ノードリスト」やそれに似た情報を配布するサーバが存在する。

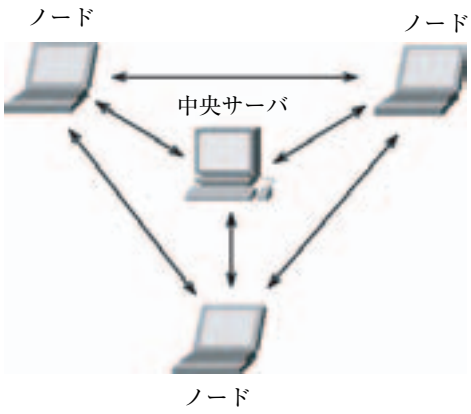

図1．Hybrid P2Pの通信形態

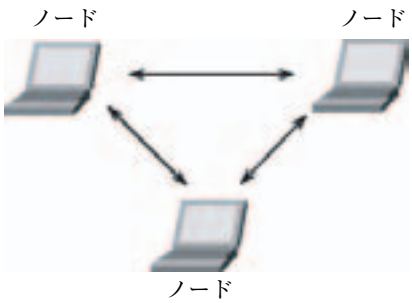

図2．Pure P2Pの通信形態

EveLiCoではユーザが気軽に利用できることを重要視しているため、中央サーバの設置の手間がいらないピュアP2Pの形態を採用した。(ただし、利用する組織内のネットワーク環境によってはハイブリッドP2Pも選択できるようにしている(8.1参照))。

2.2　トリップによるユーザ識別

一般的なチャットでは他人に成りすますことができないように、サーバがそのユーザにユニークなIDを付与するか、ユーザのIPアドレスをIDとしてチャットを行う。しかし、ピュアP2Pではサーバは存在しないため、そのネットワーク上でユニークなIDを管理/付与する事ができない。また、IPアドレスをユニークなID

として利用する場合、無線LANを利用しているユーザは接続の度にIPアドレスが変わることがあるので、長期的にユーザを証明するIDとすることができない。

このため、ユーザ自身が入力したパスワードからトリップと呼ばれる暗号化された文字列を生成し、それをユーザのIDとすることにした。トリップはパスワードが同じ場合、生成されるトリップも同じ文字列になる。よって、本研究ではなるべく他のユーザとパスワードが同じにならないように、パスワードは5文字以上でないとトリップを生成できないように制限している。

また、「aaaaaaaa」等の単純なパスワードや「hogehoge」「password」等の不適切なパスワードでもトリップを生成できないように制限している。トリップが生成できなければネットワークには参加できないので、トリップの生成は不可欠である(図3)。

図3．エントリーウインドウ

2.3 トリップの生成方法

半角英数字で最大8文字までのパスワードをKeyとする。パスワードの2文字目、3文字目をSaltとする。このKeyとSaltをcrypt関数［7］の第一引数と第二引数に渡してDES（Data Encryption Standard）暗号処理をする。暗号化された文字列の後ろから10文字を抜き出したものがトリップとなる。

参考文献［2］より、下記にトリップの生成方法を示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1  char*s=(char*)malloc(strlen(pass)+3),*t, salt[3];
2  char *tr[2] = {":;<=>?[@￥￥]^_`",
                                    "ABCDEFGabcdef"};
3  sprintf(s, "%sH.", pass);
4  sprintf(salt, "%c%c", s[1], s[2]);
5  free(s);
6  for (s=salt; *s; s++){
7          if (*s < '.' || *s > 'z') *s = '.';
8          if ((t=strchr(tr[0], *s)) != NULL)
9                  *s = tr[1][t-tr[0]];
10 }
11 s = crypt(pass, salt);
12 printf( &s[strlen(s)-10]);
```

3．通信路の暗号化

3.1 暗号化の必要性

LAN上を流れる全パケットはNIC（Network Interface Card）をプロミスキャスモード（Promiscuous Mode）に設定することで容易にキャプチャが可能なので、ある特定の相手とのプライベートなコミュニケーションを行うソフトウェアの場合、プライバシーを考慮するとパケットを暗号化することが望ましい。

3.2 OpenSSLによるBlowfish暗号・復号

本研究では公開鍵方式で共通鍵の受け渡しを行い、その後の通信は共通鍵で行うこととした。暗号/復号ライブラリとしてはOpenSSL［3］のWindows用バイナリ［4］を使用した。OpenSSLはオープンソースで無料で利用することができる。ここではOpenSSLを用いて共通鍵暗号方式のBlowfish暗号/復号について解説する。

共通鍵暗号方式ではDES（Data Encryption Standard）が有名だが鍵長が56bitと短く、現在では暗号強度が低いため、本研究では鍵長を32bitから448bitまで設定可能（本研究では128bit）なBlowfishを採用した。Blowfishで暗号/復号処理を行うために必要な鍵の生成方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1  int c;
2  unsigned char ranStr[COMMON_KEY_LEN+ 1]="";
3  BF_KEY key;
4  for (int i=0; i<COMMON_KEY_LEN; i++) {
5        while( (c=rand()%256) == 0);
6        ranStr[i]=(unsigned char)c;
7  }
8  memset(&key, 0, BF_BLOCK);
9  BF_set_key(&key, sizeof(ranStr)-1, ranStr);
```

*COMMON_KEY_LEN*には鍵のサイズをバイト単位で指定する。rand関数を用いてランダムな文字列を作成し、それを元にBF_set_key関数で鍵を生成する。

次に、Blowfishで暗号処理を行う方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1  unsigned char ivec[BF_BLOCK]="";
2  int outlen = 0;
3  BF_cbc_encrypt(rawStr, cryptStr, len, bkey,
                                  ivec, BF_ENCRYPT);
```

*rawStr*には平文のデータ、*len*にはそのデータサイズが格納されている。第6引数にBF_ENCRYPTを指定してBF_cbc_encrypt関数を実行すると、第2引数の*cryptStr*には暗号化されたデータが格納される。

次に、Blowfishで復号処理を行う方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1  unsigned char ivec[BF_BLOCK]="";
2  BF_cbc_encrypt( inStr, decryptStr, inlen, bkey,
                                  ivec, BF_DECRYPT);
```

*inStr*には暗号化されたデータ。*inlen*にはそのデータサイズが格納されている。*bkey* にはBlowfishの鍵が格納される。第6引数にBF_DECRYPTを指定してBF_cbc_encrypt関数を実行すると、第2引数の*decryptStr*には復号されたデータが格納される。

3.3　OpenSSLによるRSA暗号・復号

参考文献［5］より、OpenSSLを用いて公開鍵暗号方式のRSA暗号/復号について説明する。

RSAで暗号/復号処理を行うには、公開鍵と秘密鍵のペアを生成する必要がある。下記にその方法を示す。また、斜体の変数はあらかじめ定義された型で宣言、構築されたものとする。

```
1  // キーペアの作成
2  RSA *rsaKey = RSA_generate_key(
         RSA_KEY_LEN, RSA_F4, NULL, NULL);
3  // 公開鍵をPEM形式で書き出し
4  PEM_write_RSAPublicKey(publicKeyFile,
                                  rsaKey);
5  // 秘密鍵をPEM形式で書き出し
6  PEM_write_RSAPrivateKey(privateKeyFile,
           rsaKey,NULL, NULL, 0, NULL, NULL);
7  // 領域の開放
8  RSA_free(rsaKey);
```

生成した公開鍵と秘密鍵はPEM（Privacy Enhanced Message）形式でテキストファイルに出力する（図4）。

公開鍵で暗号化する方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言、構築されたものとする。

```
1  RSA *rsaPubKey;
2  pubKeyFile = fopen(openName, "r");
3  rsaKey=PEM_read_RSAPublicKey(pubKeyFile,
```

図4．出力されたRSAの公開鍵と秘密鍵

```
                                    NULL, NULL, NULL);
4   RSA_public_encrypt(sizeof(data), data, outbuf,
                        rsaPubKey, RSA_PKCS1_PADDING);
```

PEM_read_RSAPublicKey関数にて、出力されたPEM形式の公開鍵を読み込む。RSA_public_encrypt関数に暗号化対象のデータを指定して暗号化を行う。暗号化されたデータは*outbuf*に格納される。

秘密鍵で復号化する方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言、構築されたものとする。

```
1   RSA *rsaPriKey;
2   priKeyFile = fopen(openName, "r");
3   rsaPriKey=PEM_read_RSAPrivateKey(priKeyFile,
                                    NULL, NULL, NULL);
4   RSA_private_decrypt(sizeof(cryptData),cryptData,
        outBuf, rsaPriKey, RSA_PKCS1_PADDING);
```

PEM_read_RSAPrivateKey関数にて、出力されたPEM形式の秘密鍵を読み込む。RSA_private_decrypt関数に復号対象のデータを指定して復号を行う。復号されたデータは*outbuf*に格納される。

3.4 共通鍵暗号方式の鍵の共有

EveLiCoでは暗号/復号処理の速い共通鍵暗号方式を使って通信を暗号化しているが、共通鍵暗号方式ではどうやって安全に相手に共通鍵を渡すのかが問題となる。

これを解決する方法として、まず通信を暗号化したいノードは相手の公開鍵を使って共通鍵を暗号化する（3.3参照）。そして公開鍵とペアとなる秘密鍵を持った相手に暗号化された共通鍵を送信する。ペアとなる秘密鍵を持ったノードでなければこの共通鍵を復号することはできないので安全に共通鍵を共有することが可能である。

4．チャットルーム

4.1 チャットルームの概要

EveLiCoでは2人以上でチャットをする際に使うチャットルームと、2人だけでチャットをするフレンドチャットがあり、ここではチャットルームを実現する手法を説明する。フレンドチャットについては6.4を参照されたい。

本研究では、各ユーザが好きな部屋名のチャットルームを作ることができ（図5）、チャットルームを作ったユーザをホストと呼ぶ。ホストは部屋名のほかにその部屋の最大接続人数、入室パスワードも設定することができる。

ホストはユーザの入室管理のみを行うだけで、入室後はルーム内のユーザ同士でP2Pにて全て通信を行う（4.4）。また、ユーザは複数のチャットル

図5．チャットルーム作成ウインドウ

ームを作成することができ、複数のチャットルームに同時に入室することができる。

4.2　チャットルーム検索

ユーザはネットワーク上にある全てのチャットルームを検索し、それらのルーム情報を収集することでルームを一覧表示することができる。ユーザはチャットルーム名を見て、興味のある部屋に入室することができる。これにより、共通の話題を持ったユーザ同士が集まりやすいようになっている。

4.2.1　検索手順

マルチキャストとは、1対多の通信を行う通信方法である。ブロードキャストは不特定多数のコンピュータにパケットを送信するのに対し、マルチキャストではある特定の複数のコンピュータだけにパケットを送信することができる（図6）。

ノードがマルチキャストでパケットを受信するには、特定のマルチキャストグループに参加する必要があり、またルータには参加したノードのIPアドレスが保存されていく。ノードがマルチキャストパケットを送信する場合は特定のマルチキャストグループ宛にパケットを送信する。ルータはこのパケットを受け取ると、マルチキャストグループに保存されているIPアドレス群にパケットを複製して送信する。

EveLiCoでは通信ライブラリにWinSockを使用しており、参考文献［6］よりWinSockでマルチキャストを行うためのUDPソケットの作成方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言、構築されたものとする。

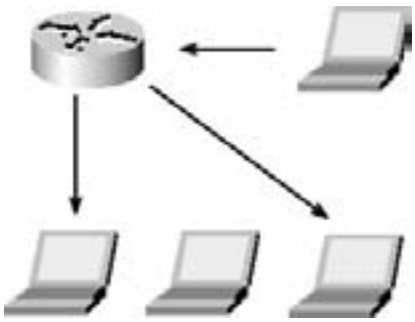

図6．マルチキャスト

```
1  SOCKET so,
2  SOCKADDR_IN mySA;
3  int nRtn;
4  //  UDPソケットの作成
5  memset(&mySA, 0, sizeof(SOCKADDR _IN));
6  mySA.sin_family = AF_INET;
7  mySA.sin_port = htons(port);
8  mySA.sin_addr.S_un.S_addr = INADDR_
   ANY;
9  so = socket(AF_INET, SOCK_DGRAM, 0);
10 if(so == INVALID_SOCKET){
11       /* エラー処理 */
12 }
13 nRtn=WSAAsyncSelect(so, hWnd, MSG_ UDP,
                                  FD_READ);
14 if(nRtn == SOCKET_ERROR){
15       /* エラー処理 */
16 }
17 nRtn = bind(so, (struct sockaddr)mySA,
                        sizeof(SOCKADDR_IN));
18 if(nRtn == SOCKET_ERROR){
19       /* エラー処理 */
20 }
```

次に、UDPソケットにマルチキャストソケットの設定を行う。下記にその方法を示す。斜体の変数はあらかじめ定義された型で宣言、構築されたものとする。

```
1  //  ソケットにマルチキャストを設定
2  struct ip_mreq mreq;
3  memset(&mreq, 0, sizeof(mreq));
4  mreq.imr_interface.S_un.S_addr=INADDR _ANY;
5  mreq.imr_multiaddr.S_un.S_addr =
                     inet_addr(MULTICAST_IP);
6  if (setsockopt(so, IPPROTO_IP,
              IP_ADD_MEMBERSHIP,
              (char *)&mreq, sizeof(mreq)) < 0){
```

```
7        /* エラー処理 */
8   }
```

*MULTICAST_IP*にはマルチキャストグループのアドレスを指定する。

次に、UDPソケットにIP_MULTICAST_LOOPを設定する方法を示す。これはノードがマルチキャストでパケットを送信した際、ルータから送信ノード宛てにパケットが送信されるのを防ぐ設定である。

下記に設定方法を示す。斜体の変数はあらかじめ定義された型で宣言、構築されたものとする。

```
1   char loopch=0;
2   if(setsockopt(so, IPPROTO_IP,
                    IP_MULTICAST_LOOP,
              (char *)&loopch, sizeof(loopch)) < 0){
3       /* エラー処理 */
4   }
```

EveLiCoでは、全てのノードは特定のマルチキャストグループに必ず参加する。ノードがチャットルームを検索する場合、このマルチキャストグループ宛にRoomRequestパケットを送信する。RoomRequestパケットはルータによりマルチキャストグループに所属する全ノードに送信され、RoomRequestパケットを受け取ったノードがホストならば自身が管理しているチャットルーム情報（部屋名、接続人数、パスワードの有無）を、RoomRequestパケットを送信したノードに返信する。このパケットはRoomInfoパケットと呼ぶ。

ホストから送られてきたRoomInfoパケットに格納されているルーム情報は、リスト構造によって管理され、受信したルーム情報の数だけ動的にメモリを確保して保存する。

受信したルーム情報はリストビューに一覧表示される（図7）。

4.3 入室処理

図7．チャットルームの検索結果の画面

ノードはネットワーク上のルーム情報を取得後、ルーム名の一覧を見て興味のあるチャットルームに入室することができる。下記に入室手順を説明する。

（1）ホストとの接続処理

ノードは入室したいチャットルームを見つけたら、そのチャットルームを作成したホストとTCP/IPにて接続する。接続後は通信を暗号化するために3.4で説明した方法で共通鍵をホストと共有する。共通鍵は接続処理の度にランダムに生成する（3.2参照）。さらに、ユーザ名とトリップも両者の間で交換する。

（2）入室可否の判定

接続処理後、ノードはホストにルーム参加要求パケットを送信する。ホストはこのパケットを受け取ったら以下の3つのチェックを行う。

1. ルーム内の接続人数に空きがあるかどうかを調べる。
2. 参加要求パケットの送信元のIPアドレスから、このノードが既に同一ルームに参加しているかどうかを調べる。
3. パスワードが設定されているルームの場合、参加要求パケットにはノードが入力したパスワードが保存されている。このパスワードが正しいものか調べる。

以上の3つのチェックを全てパスした場合、ホストは入室許可パケットを入室申請中のノードに送信する。もし１つでもパスできなかった場合、入室不許可パケットを入室申請中のノー

ドに送信する。この入室不許可パケットには不許可になった理由を示す番号が保存されている。ノード側ではその番号に対応したメッセージを表示することで、なぜ入室できなかったのかを知ることができる。

（3）ホスト以外の他ノードとの接続処理

ノードは入室許可パケットを受信後、ホストからルーム内の全ノードのIPアドレスとポート番号を受信し、(1) と同じ方法でこれらのノードと接続処理を行う。

4.4　チャットの手順

入室後はチャットをすることができ、テキストボックスに発言内容を入力し、Enterキーを押すことで発言できる（図8）。また、画面右上の接続人数ボタンをクリックすると接続者の名前の一覧が表示される。

入室時に既にルーム内の全ノードと個別の共通鍵を共有しているので、この共通鍵で発言内容を格納したパケットを暗号化（3.2参照）して全ノードに送信する。受信側ではこのパケットを復号化し、ログを表示するテキストボックスに発言内容を表示させる。

また、ユーザがチャットをしている間、周囲の者から画面を覗かれてもユーザのプライバシーが守れるように、チャットの文章を読みにくくするためにチャットウインドウの半透明化も可能である（図9）。

ウインドウの半透明化を行う方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1　SetLayeredWindowAttributes(hWnd, 0,
                    128, LWA_ALPHA);
```

第3引数で透明度を表す。値が小さい程、ウインドウが透明になる。設定できる値は0〜255である。

4.5　退室処理

ユーザがチャットウインドウを閉じるとルーム内の全ノードとの接続を終了する。TCP/IP

図8．実際のチャットの様子

図9．チャットウインドウの半透明化

の接続を正常に終了する方法を下記に示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1　shutdown(sock, SD_SEND);
2　while(true){
3　　n = recv( sock, buf, sizeof(buf), 0);
4　　if(n == 0 || n == SOCKET_ERROR) break;
5　}
6　shutdown(sock, SD_BOTH);
```

7 closesocket(*sock*);

切断処理を開始する側でshutdown関数にSD_SENDを指定して呼び出すことで、切断処理を開始する側ではこれ以上パケットを送信できないように設定する。次に、相手から送信されるデータがあれば全て受信し、shutdown関数にSD_BOTHを指定して呼び出し、相手側からのパケットを受信しないように設定する。最後にclosesocket関数でソケットを解放する。

5．ファイル送受信

5.1 ファイルの公開

チャットルームに参加している場合、そのチャットルーム内の全ノードに対し、指定したファイルを公開することができる。ファイルを公開するには、チャットウインドウにファイルをドラッグ＆ドロップする。すると受信側のチャットウインドウには［Click］というリンクが表示され、リンクをクリックするとファイルの転送が開始される。(図10)

一度に複数のノードと複数のファイルの送受信が可能であり、ファイルの転送状況はプログレスバーによるファイルの転送状況と転送速度がチャットウインドウに表示される（図11)。

ファイルの公開､送受信の手順を下記に示す。

（1）ファイル情報の取得

ユーザはチャットルームに、公開したいファイルをチャットウインドウにドラッグ＆ドロップするとファイル名、ファイルサイズ、ハッシュ値（5.3参照）を取得することができる。

（2）ファイル情報の公開

ファイル情報の取得後、ファイルを公開するノードはチャットルーム内の全ノードにファイル情報（ファイル名、ファイルサイズ）を送信する。

（3）接続処理

ファイル情報を受信したノードのチャットウインドウには、ファイル公開者名、ファイル名、ファイルサイズ等が表示される。

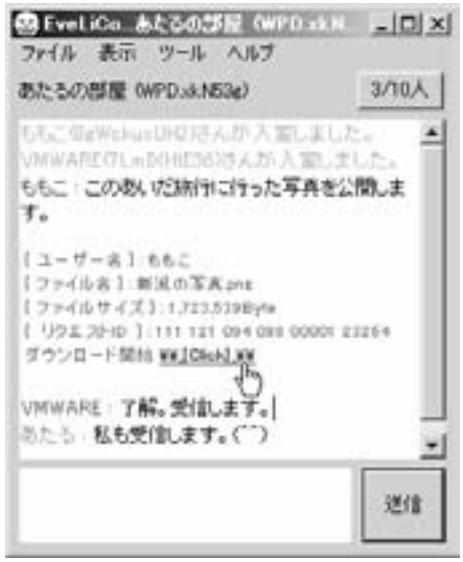

図10．ファイルの公開の様子

図11．ファイル転送の様子

さらに、チャットウインドウに［Click］と書かれたリンクを表示する。このリンクをクリックすることでファイル公開者とTCP/IPで接続を開始する。

（4）ファイル情報の転送

(2) で受信したファイル情報は、チャットウインドウに書き込まれるだけでメモリ上には保存されない。TCP/IP接続処理後、受信ノード

は送信ノードから詳細なファイル情報（ファイル名、ファイルサイズ、ファイル識別ID）を受信する。

（5）スレッドによる送受信処理

両ノードは送信処理と受信処理を別スレッドにて行う。これにより、チャットの処理には影響を与えずに送受信が可能である。

また、チャットルーム内の特定のノードだけにファイルを送信したい場合は、7.1で説明する簡易メッセージ機能を使うことで可能となる。

5.2　レジューム機能

レジューム機能とはファイルの転送中に送信ノードと受信ノードのどちらかがファイルの転送を中断しても、同一ファイルを再び転送するとファイルの中断された部分から転送を開始できる機能である

下記にその手順を解説する。

（1）ファイルのハッシュ値を求める

ファイル送信前に送信ファイルのハッシュ値を求め（5.3参照）、ファイル名やファイルサイズと共に受信ノードに送信する（図12）。

（2）ハッシュ値の保存

受信ノード側は送られてきたハッシュ値をローカルファイルに保存する。

（3）ファイル転送の開始

ファイル転送を開始し、受信ノードは受信しているファイルデータをローカルファイルに保存していく。

（4）ファイル転送の中断

何らかの理由でファイル転送が中断されると、受信していたファイルは未完成のままローカルファイルに保存されることになる。

（5）ファイル転送の再開

（1）、（2）と同じ方法でファイル転送を再開するが、この時受信ノードは送信ノードから送られてきたファイルのハッシュ値が既にローカルファイルに保存されているかを調べる。受信が完了したファイルのハッシュ値は削除されるため、もし同一のハッシュ値がローカルファイルに残っていれば、そのハッシュ値のファイルは過去に中断されたものであると判断できる。

図12．ハッシュ値の計算

（6）ファイルサイズを送信する

受信ノードは中断されたファイルのファイルサイズを送信する。受信ノードは中断されたファイルをオープンして待機する。

（7）ファイルの途中からの転送

送信ノードは中断されたファイルサイズからファイルを読み込み、受信ノードに転送開始する。

5.3　ハッシュ値の生成

ハッシュ値はデータから得られる疑似乱数であり、データが1ビットでも変化すると全く別のハッシュ値が生成される。この特性を利用し、例えばファイル名とファイルサイズが全く一緒のファイルでもハッシュ値によってファイルの識別が可能である。EveLiCoではファイルのレジューム機能（5.2参照）でハッシュ値によるファイルの識別を行っている。

ハッシュ値を求める方法としてSHA256（Secure Hash Algorithm 256）を採用しており、下記にOpenSSLでSHA256のハッシュ関数を利用する方法を示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1　fp = fopen(fileName, "rb");
```

```
2  SHA256_CTX ctx;
3  SHA256_Init(&ctx);
4  //  ファイルからハッシュ値の計算
5  while( true ){
6      nLen = fread( buf, 1, sizeof(buf), fp);
7      if(nLen == 0){
8          if( 0 == GetLastError() )
9              break;
10         else
11             return -1;
12         }
13     SHA256_Update(&ctx, buf, nLen);
14     }
15     //  ハッシュ値を取得
16     SHA256_Final(&sha[0], &ctx);
17     //  ハッシュ値を16進数で表示
18     for(int i = 0; i < SHA256_DIGEST_LENGTH;
       i++) {
19     sprintf ( &ascii[i*2],"%02x", sha[i]);
20     }
21 }
```

本研究では、SHA256で生成したハッシュ値（256bit）を16進数で表示している。得られるハッシュ値は固定長で64文字となる。

5.4 ファイル送信パケットの暗号化

オプションとして、ファイル送信のパケットを共通鍵で暗号化して送信することができる。ファイルの送受信時に発生する膨大な数のパケットを盗聴・構築することは難しいため、ファイルの送受信においてパケットを暗号化する必要性は低いと考えられる。さらに、パケットを暗号/複号するにはコンピュータのCPU（Central Processing Unit）に負荷がかかり、処理に時間がかかるためファイルの転送効率は悪くなり、CPUによっては8割程度まで通信速度が落ちる。

しかし、機密性の高いファイルを送信する場合等、ユーザがファイルを暗号化して送信したいと判断する場合も考えられるため、暗号化の有効/無効を選択できるようにした。

6．フレンド

6.1 フレンド機能の概要

チャットルームで出会った相手をフレンド登録することが可能であり、フレンド登録した相手をフレンドと呼ぶ。フレンドとはいつでもチャットや簡易メッセージ（7.1参照）を使うことができる。

具体的には、メインウインドウのツリービューにはフレンドの名前とトリップが表示され（図13）、オンライン状態のフレンドの名前をダブルクリックするとそのフレンドと1対1のチャットが可能である。さらに、メニューから［ツール］、［簡易メッセージを送信する］の順に項目をクリックすることで、フレンドとは7.1で説明する簡易メッセージを使うことができる。

6.2 フレンド登録

フレンド登録する場合、チャットウインドウのメニューから項目［フレンドに追加する］をクリックすることでフレンド申込みウインドウが表示される（図14）。

ノードはチャットルーム内の相手にフレンドを申込むと、相手にフレンド申込みの簡易メッセージ（7.1参照）が自動的に送信される。簡易メッセージを受け取った側は、申込者の名前とトリップを見て、フレンド登録を許可するかどうか選択できる（図15）。

フレンド登録を許可した場合、両者の間で名前とトリップ、そして両者の公開鍵を交換する。ここで交換する公開鍵は、ユーザが初回にトリップを生成した時に一度だけ生成したもので、ペアの秘密鍵と共にローカルファイルに保存され、常に使用する。両者の間で交換された公開鍵は6.3で説明するフレンド認証で使用することになる。

6.3 オンライン通知

フレンドがオンライン状態になった場合、タスクトレイにバルーンが表示されるようになっている（図16）。バルーンを表示することで、フレンドが通信可能な状態になったことをすぐ

図13．フレンドのオンライン状況表示

図14．フレンド登録申請ウインドウ

に知ることができる。

　下記にタスクトレイにバルーンを表示する方法を示す。斜体の変数はあらかじめ定義された型で宣言されたものとする。

```
1  NOTIFYICONDATA ni;
2  memset(&ni, 0, sizeof(NOTIFYICONDATA));
3  ni.cbSize = sizeof(NOTIFYICONDATA);
4  ni.hWnd = hWnd;
5  ni.uID = ID_MYTRAY;
6  ni.uFlags = NIF_ICON | NIF_MESSAGE |
                              NIF_TIP | NIF_INFO;
7  ni.hIcon = (HICON)LoadImage(g_hInst,
                MAKEINTRESOURCE(IDI_ICON),
                IMAGE_ICON, 16, 16, 0);
8  strcpy(ni.szTip, "ツールチップ");
9  ni.uCallbackMessage = MYTRAY_MESSAGE;
10
11 //バルーンチップを表示するための設定
12 ni.uTimeout = 10000;
13 ni.dwInfoFlags=NIIF_INFO | NIIF_NOSOUND;
14 lstrcpy(ni.szInfoTitle, "バルーンタイトル");
15 wsprintf(ni.szInfo, "バルーンの内容",
                                  (LPSTR)lParam);
16 Shell_NotifyIcon(NIM_DELETE, &ni);
17
18 // タスクトレイにアイコンを追加
19 if( Shell_NotifyIcon(NIM_ADD, &ni) == false){
20       /* エラー処理 */
21 }
```

図15．フレンド登録の申込みメッセージ

図16．フレンド登録申請ウインドウ

次に、マルチキャストを用いてサーバや初期ノード情報を使わずに、フレンドのオンライン状態を知る方法を下記に示す。

(1) あるノードがオンラインになった時、そのノードは自身のトリップをマルチキャスト通信で全ノードに送信する。

(2) トリップを受信したノードは、自身のフレンドリストの中に一致するトリップがあるか調べる。

(3) フレンドリストの中に一致するトリップが無い場合は、受信したトリップの持ち主とはフレンドでは無いので何もしない。もし一致するトリップがある場合は自分のトリップを送信し、オンラインになったノード側にも自分のトリップが登録されているか確認してもらう。

(4) 両者がトリップを確認後、オンラインになったノード側から相手の公開鍵を用いてフレンド認証を行う。フレンド認証を行うことで、偶然トリップが一致した別人ではないかどうかを調べる。

(5) 両者の間でフレンド認証が正しく完了できたら、タスクトレイにバルーンを表示する。

6.4 フレンドチャット

フレンド認証の完了後、メインウインドウのツリービューにフレンドの名前が表示され、相手の名前をダブルクリックすることで、1対1のチャットを行うことができる。これにより、チャットルームを作成、検索する手間をかけずにフレンドとチャットをすることができる。

7. 簡易メッセージ

7.1 簡易メッセージ機能の概要

簡易メッセージとは、ファイルやテキストメッセージを送信することができる機能である。簡易メッセージの送信が可能な相手は、同じチャットルーム内にいるノードと、フレンド認証済みのノードである。これらのノードとは既に接続済みであるので、その接続を使って送信相手、送信メッセージ、送信ファイルを指定し簡易メッセージを送信する（図17）。

図17．簡易メッセージ送信ウインドウ

図18．簡易メッセージの受信通知

簡易メッセージ機能を使うことで、例えばオンライン状態のフレンドに対して短い言付けがある場合、チャットを使わずにメッセージを伝えることができる。また、チャットルーム内の誰か一人に対してファイルを送信したい場合でも、簡易メッセージを使うことで可能となる。

また、簡易メッセージを受信した側のタスクトレイにはバルーンが表示され（図18）、そのバルーンをクリックすることでメッセージボックスを開くことができる。

7.2 メッセージの送受信

簡易メッセージ機能では他ノードに対して

1KB以内のテキストメッセージを送信可能である。

送信したメッセージは相手のローカルファイルに保存され、メッセージボックスウインドウでいつでも見ることが可能である（図19）。メッセージにファイル情報が添付されている場合、『ファイルを受信する（1）』ボタンが表示される。

7.3　ファイルの送受信

簡易メッセージ機能では他のノードに対してファイルの送信が可能である。

手順としてはまず、突然ファイルの送信を開始するのではなく、最初は送信者の名前とファイルの情報（ファイル名、ファイルサイズ）を簡易メッセージにて送信する。次に、受信側のノードはそれらの情報を見て受信するかどうかを判断し、受信する場合はメッセージボックスウインドウにある『ファイルを受信する』ボタン（図19の（1））をクリックするとファイルの受信が開始される。また、ファイルの送受信状況はメッセージボックスウインドウにあるタブに表示される（図20）。送受信のタスクごとに、通信相手、ファイル名、転送状況のパーセンテージ（送信タスクはオレンジ色、受信タスクは青色）、完了までの残り時間、転送速度等が表示される。

受信側のノードがファイルを受信するかどうかを判断できるようにすることで、送信側からの一方的なファイル送信をできないようにした。

図19．簡易メッセージのメッセージボックス

図20．ファイル転送タブを開いた状態

8．サーバ

8.1　サーバの概要

EveLiCoではマルチキャストを使うことで、サーバがなくともLAN内の特定の複数のノードにパケットを送信することができる。

しかし、組織内に複数のLANがあり、さらにそれらのLANの間でマルチキャストルーティングが設定されていない場合、他のLANにいるノードにパケットを送信することはできない。

こういった環境下で、なおかつ他のLANのノードとも通信を行いたい場合は、ルータにマルチキャストルーティングを設定することで通信が可能である。しかし、ルータの設定を変えるためには、その組織でのネットワーク管理権限と専門的な知識と技術が必要である。

そこで、管理サーバを使ったハイブリッドP2Pで通信させることで、ルータの設定を変更すること無く、複数のLAN間でのパケット送信を可能にした。

また、サーバソフトはWindowsバージョンとLinuxバージョンを開発した。Windowsバージョンでは GUI（Graphical User Interface）、

図21．サーバソフトのウインドウ

図22．サーバソフトのアイコン化

図23．Linux版サーバのインストールと実行

LinuxバージョンではCUI（Character-based User Interface）として作られているが、使用する通信プロトコルは同じである。

8.2 サーバ（Windowsバージョン）

Windowsバージョンのサーバでは、サーバソフトを起動後、『開始』ボタンをクリックすることで、サーバを開始できる（図21）。このサーバはノードのIPアドレスとPort番号を管理する。

また、サーバソフトは常に起動している必要があるので、邪魔にならぬようにタスクトレイにアイコン化できるようにし、右クリックでメニューも表示するようにした。（図22）

8.3 サーバ（Linuxバージョン）

Linuxバージョンのサーバソフトはmakeコマンドによりインストールすることができる（図23の（1））。また、make cleanコマンドによりアンインストールすることができる。

実行は #./eveserver［待機Port番号］ で行う（図23の（2））。

8.4 ノード側の設定

ハイブリッドP2Pで通信を行うには、ノード側にサーバのIPアドレスとport番号を設定する必要がある。

設定するには、メインウインドウのメニューからネットワーク設定画面を開き、そこでサーバのIPアドレスとport番号を入力する（図24）。

また、組織内のサーバのIPアドレスとPort番号が書かれた設定ファイル（図25）を、あらかじめノード側のソフトに同梱しておくこともできる。組織内でサーバを設置する人が、設定ファイルの同梱されたソフトをその組織内で配布することで、ユーザは何も設定することなくハイブリッドP2Pでの通信が可能となる。

8.5 通信手順

サーバがマルチキャストパケットの届かない他のLANへパケットを送信する手順を説明する。

（1）ノード情報の登録

ノードは起動時にサーバへ自分のIPアドレスとPort番号をUDPで送信し、サーバはそのノードのIPアドレスとPort番号を保持しておく。

（2）マルチキャストパケットの送信

ノードは必要に応じてマルチキャスト通信で

図24．ノード側のネットワーク設定ウインドウ

図25．同梱する際の設定ファイル（evelico.ini）

同じLAN内の他ノードにパケットを送信している(ルーム検索、フレンドのオンライン通知)。この時、サーバにも同一のパケットをUDPで直接送信する。

（3）マルチキャストパケットの転送

サーバはパケットを受信し、保持している全ノード宛てにUDPで直接パケットを送信する。

UDPで直接送信することで、マルチキャストでは届かない他ノードにもパケットを送信することができる。

9．実証実験と考察

9.1　実験目的と実験環境

今回開発した EveLiCo が当初設定した目的の機能と性能を持っているかどうかを検証するために実証実験を行った。評価すべき項目としては、

- ・学内全域から有線無線を問わず通信可能かどうか
- ・各ノードのソフトのメモリ使用率、CPU使用率が十分に低いかどうか
- ・実装した各機能を使用した際のバク洗い出し
- ・ファイル転送機能の転送速度はどの程度か
- ・実験参加者に詳細な使用方法を教えなくても使用可能かどうか

である。

実験環境としては、6名の参加者で7台のWindowsマシンを使用し、OSはVistaが5台、XPが2台である。また、マルチキャストルーティングが設定されていないLAN間でも通信が行えるかどうかも確認するため、マルチキャストルーティングが設定されていない大学内の3ヶ所のLANに2名ずつ割り当てた。また、内1台はノートパソコンで無線LANにて実験を行った。

9.2　実験結果と考察

東京情報大学内幹線ネットワークのルータはマルチキャスト機能が使用可能には設定されていないため、大学内の全ネットワークを対象としたP2Pネットワークの構築には簡易サーバ(8.1参照)を設置して実証実験を行った。

チャット時には特定のノードと通信ができないバグが発生した(その後修正した)。その他のフレンド登録、ファイル転送、簡易メッセージの送受信は正常に動作確認できた。

7台でチャットを行った各ノードのメモリ使用量は10MB程度(図26)で、CPUの使用率はチャット時には0～3%であり、CPU、メモリ負荷の観点からは十分快適な使用感であると言える。(図27)。ただし、ファイル転送時は10～30%となり、また物理帯域速度100Mbpsの環境でファイル転送速度は非暗号で58Mbpsであった。

実験参加者達はチャットルームを自由に作成、検索できることでユーザ主体のネットワー

図26．本研究のソフトウェアのCPU/メモリ使用量

図27．本研究のソフトウェアのCPU使用率履歴

クを構築することができた(図28)。

また、今回はノートパソコンで大学内の無線LANでの実験も行い、無線LANでのネットワーク参加も動作確認ができた。これにより、大学全域にある無線LANを使うことで、サーバなしで大学内のどこからでもネットワークに参加できることが確認できた。

今回の実験ではP2Pによる利便性が確認でき、不具合や問題等は発生しなかった。しかしユーザ同士のコミュニケーションの秘匿性を重視しチャット等の通信内容は暗号化しているため、今後利用者が増えると特定の人物に対する誹謗中傷や、著作権を無視したファイルの転送が行われる危険性も無視できないものであり、開発にあたる上でもこれらの対策は今後の課題であると思われた。

10．まとめ

本研究ではマルチキャストとP2Pによって気軽にコミュニケーションを行える方法を提案し、その方法を実装したソフトウェアEveLiCoを開発した。マルチキャストを使うことにより、初

図28．実証実験時のチャットの様子

期ノード情報の配布が不要なピュアP2Pソフトとして機能する。全てのコミュニケーションの通信は暗号化されているため盗聴の危険性がない。また、匿名性が保障されている一方で成りすましの危険性があるため、トリップ機能の実装によりこれをほぼ不可能とした。さらにチャットだけではなく1対1のコミュニケーションやファイル転送機能も持たせたことにより、ユーザ同士が文字情報以外でもコミュニケーションを促進できるようにした。これらによって当初目標とした全ての機能の実装が完了し、実験により、組織内のLAN内でサーバなしでネットワークを構築することができ、距離的に近い人間同士で気軽にコミュニケーションをとることができることがわかった。

現代社会のコミュニケーションをとることが苦手な若者同士でも、コンピュータとネットワークを使った匿名性のあるコミュニケーションならば、比較的容易に交流を深めることができるであろう。EveLiCoは組織内のネットワークを利用するため、仮想的なコミュニケーションを通して親しくなった相手とは、実際に会って交流を深めることもできる。また、既に親しい者同士でも、チャットはもちろん、簡易メッセージでメッセージのやり取りや、他のコンピュータへ気軽にファイルを送信することもできる。

EveLiCoの導入に際しては、複数のLAN同士

でマルチキャストルーティングがされている組織内であればサーバも不要なため、個々のコンピュータにEveLiCoをインストールするだけでネットワークを構築できる。EveLiCoは設計上、実際に通信を行うのはチャットルーム内のノードと、フレンドのノードだけであるため、ネットワーク上に数百、数千のノードがいても個々のコンピュータには負荷はかからず、大規模なネットワークを構築することも可能である。

このソフトは筆者が自分の大学内の学生同士のコミュニケーションを円滑にしたいと考えて開発したものだが、近年では企業内での社員同士のコミュニケーション不足も指摘されているため、大学だけでなく企業のオフィスやその他の組織等でも利用し、コミュニケーションを円滑に行うために役立ててほしいと考えている。

【参考文献】

[1] 金子勇：Winnyの技術，アスキー（2005）
[2] 2ch, *Make of Trip* at http://nssearch.hp.infoseek.co.jp/clang/1035211039.html
[3] OpenSSL, *Manpage of OpenSSL* at http://www.openssl.org/
[4] Shining Light Productions, *Manpage of Win32 OpenSSL* at http://www.slproweb.com/index.html
[5] Fire Project, *Manpage of OpenSSL RSA* at http://www.fireproject.jp/feature/c-language/openssl/rsa.html
[6] Geekなページ, *Make of UDP Socket* at http://www.geekpage.jp/programming/winsock/multicast.php
[7] kenjinet.s26.xrea.com, *Make of crypt()* at http://kenjinet.s26.xrea.com/crypt.txt

謝辞

暗号化の部分で貴重なご意見を下さった、東京情報大学総合情報学部井関文一教授に感謝の意を表します。